МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ

УНИВЕРСИТЕТ им. Р.Е.АЛЕКСЕЕВА

Институт радиоэлектроники и информационных технологий

Кафедра «Информатика и системы управления»

ОТЧЕТ

**Лабораторная работа №2**

По дисциплине: «Технологии разработки ПО»

Тема: «Работа над системой Наблюдатель – Слушатель в рамках задачи чтения данных с клавиатуры компьютера»

РУКОВОДИТЕЛЬ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Чернобаев И.Д.

СТУДЕНТ:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Дубков И.А.

Группа М23-ИВТ-1

Работа защищена «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_

Нижний Новгород 2024

Цель работы:

Инструменты: OS Linux, ReactiveExtensions (для любого удобного языка).

Реализовать трекер событий клавиатуры (какая клавиша, какой событие с клавишей произошло).

Трекер запустить в отдельном потоке.

Реализовать паттерн "Наблюдатель + Итератор":

* Каждое событие трекер передает всем подписанным "слушателям" (on\_next).
* Подписчики должны отобразить полученную информацию в файл.
* Реализовать завершение работы трекера по вводу определенной комбинации клавиш (on\_complete).
* Реализовать обработку ошибок (on\_error).

Ход работы:

Для выполнения данной программы было составлено два класса: Observer и WritingSubscriber. Эти два касса служат основой для реализации модели "Наблюдатель + Итератор", где несколько слушателей могут получать информацию о событиях от наблюдателя:

class Observer:  
 def \_\_init\_\_(self, subscribers):  
 self.subscribers = subscribers  
  
 def on\_next(self, value, event\_type):  
 for subscriber in self.subscribers:  
 subscriber.on\_next(value, event\_type)  
  
 def on\_completed(self):  
 for subscriber in self.subscribers:  
 subscriber.on\_completed()  
  
 def on\_error(self, e):  
 for subscriber in self.subscribers:  
 subscriber.on\_error(e)

class WritingSubscriber:  
 def \_\_init\_\_(self, thread\_id, filename):  
 self.thread\_id = thread\_id  
 self.filename = filename  
 self.key\_count = 0  
  
 def on\_next(self, value, event\_type):  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")  
 log\_entry = f"Thread {self.thread\_id} - {current\_time}: Key {event\_type}: {value}\n"  
 with open(self.filename, 'a') as file:  
 file.write(log\_entry)  
  
 self.key\_count += 1  
 print(log\_entry.strip())  
  
 def on\_completed(self):  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")  
 log\_entry = f"Thread {self.thread\_id} - {current\_time}: detection finished\n"  
 with open(self.filename, 'a') as file:  
 file.write(log\_entry)  
 print(log\_entry.strip())  
  
 def on\_error(self, e):  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")  
 log\_entry = f"Thread {self.thread\_id} - {current\_time}: error detected:\n\t{e}\n"  
 with open(self.filename, 'a') as file:  
 file.write(log\_entry)  
 print(log\_entry.strip())

Таким образом, мы можем при помощи класса Observer инициировать проход по всем подписчикам (у которых может быть своя собственная реализация методов on\_next, on\_completed on\_error), которые уже выполняют необходимые инструкции.

Принцип, по которому мы вызываем те или иные функции, заключен в функции monitor\_keys:

def monitor\_keys(observer, key\_combination\_tracker):  
 is\_running = True  
  
 while is\_running:  
 events = keyboard.read\_event()  
 if events:  
 key = events.name  
 if events.event\_type == keyboard.KEY\_DOWN:  
 event\_type = 'pressed'  
 key\_combination\_tracker.add\_key(key)  
 elif events.event\_type == keyboard.KEY\_UP:  
 event\_type = 'released'  
 key\_combination\_tracker.remove\_key(key)  
 else:  
 continue  
  
 if key\_combination\_tracker.is\_combination\_pressed():  
 is\_running = False  
 observer.on\_completed()  
 continue  
  
 observer.on\_next(key, event\_type)  
  
 if key in BAD\_KEYS and events.event\_type == keyboard.KEY\_DOWN:  
 observer.on\_error(f"Error: Bad key '{key}' pressed.")

Для работы с данной задачей мы установили следующие правила:

* Если зажатая клавиша образует вместе с другими зажатыми клавишами нужную для выхода из программы комбинацию, то вызываем метод on\_completed.
* Если зажатая клавиша входит в определенную группу «плохих» клавиш, то мы вызываем on\_error и даем ему на вход соответствующее сообщение об ошибке
* В любом случае мы передаем информацию о нажатой клавише при помощи метода on\_next всем итераторам

Отдельно стоит отметить метод KeyCombinationTracker, который отвечает за отслеживание текущих нажатых клавиш:

class KeyCombinationTracker:  
 def \_\_init\_\_(self, required\_keys):  
 self.required\_keys = set(required\_keys)  
 self.pressed\_keys = set()  
  
 def add\_key(self, key):  
 self.pressed\_keys.add(key)  
  
 def remove\_key(self, key):  
 self.pressed\_keys.discard(key)  
  
 def is\_combination\_pressed(self):  
 return self.required\_keys.issubset(self.pressed\_keys)

Заметим, что нигде в нашей программе кроме класса WritingSubscriber мы не указываем, что из себя должны представлять подписчики наблюдателя, кроме того, что они долнжны иметь некоторые методы называемы on\_next,on\_complete,on\_error. Так, если мы создадим рудиментарный класс ErrorSubscriber, в котором не будет никаких инструкций, но который будет отличен по содержанию от WritingSubscriber, мы все равно сможем его подписать на рассылку от наблюдателя:

class ErrorSubscriber:  
 def \_\_init\_\_(self, thread\_id):  
 self.thread\_id = thread\_id  
 self.error\_count = 0  
  
 def on\_next(self, value, event\_type):  
 pass  
  
 def on\_completed(self):  
 # print(f"ErrorSubscriber {self.thread\_id}: Monitoring completed.")  
 # print(f"Total error count: {self.error\_count}")  
 pass  
  
 def on\_error(self, e):  
 # self.error\_count += 1  
 # print(f"ErrorSubscriber {self.thread\_id}: Monitoring error occurred: {e}")  
 pass

Собрав всё это вместе, получаем основу для системы. В main мы инициируем несколько итераторов и подписываем их к наблюдателю, а затем открываем функцию отслеживания клавиш в новом потоке:

def main():  
 output\_file = "keys0.txt"  
 with open(output\_file, 'w') as f:  
 f.write('Starting logging:\n')  
  
 writing\_subscriber1 = WritingSubscriber(thread\_id=1, filename=output\_file)  
 writing\_subscriber2 = WritingSubscriber(thread\_id=2, filename=output\_file)  
 error\_subscriber = ErrorSubscriber(thread\_id=3)  
  
 key\_combination\_tracker = KeyCombinationTracker(REQUIRED\_KEYS)  
  
 subscribers = [writing\_subscriber1, writing\_subscriber2, error\_subscriber]  
 observer = Observer(subscribers)  
  
 print(f"Программа запущена. Для завершения нажмите {' + '.join(REQUIRED\_KEYS)}")  
  
 key\_thread = threading.Thread(target=monitor\_keys, args=(observer, key\_combination\_tracker))  
 key\_thread.start()  
  
 key\_thread.join()  
  
 print("Программа завершена.")

В результате работы программы получаем txt файл со следующим содержимым:

Starting logging:  
Thread 1 - 2024-12-26 07:20:56.406297: Key pressed: a  
Thread 2 - 2024-12-26 07:20:56.406297: Key pressed: a  
Thread 1 - 2024-12-26 07:20:56.538046: Key released: a  
Thread 2 - 2024-12-26 07:20:56.538563: Key released: a  
Thread 1 - 2024-12-26 07:20:57.749464: Key pressed: z  
Thread 2 - 2024-12-26 07:20:57.749982: Key pressed: z  
Thread 1 - 2024-12-26 07:20:57.750506: error detected:  
 Error: Bad key 'z' pressed.  
Thread 2 - 2024-12-26 07:20:57.750506: error detected:  
 Error: Bad key 'z' pressed.  
Thread 1 - 2024-12-26 07:20:57.878881: Key released: z  
Thread 2 - 2024-12-26 07:20:57.879401: Key released: z  
Thread 1 - 2024-12-26 07:20:59.296704: Key pressed: ctrl  
Thread 2 - 2024-12-26 07:20:59.297221: Key pressed: ctrl  
Thread 1 - 2024-12-26 07:20:59.449991: detection finished  
Thread 2 - 2024-12-26 07:20:59.450499: detection finished

Таким образом, задача успешно выполнена.

Вывод:

Написание данной программы помогло мне понять основные принципы и направления работы с программированием с использованием паттерна "Наблюдатель + Итератор", а также позволило вспомнить основные принципы использования многопоточного программирования на языке python.

Приложение А – полный код программы

import threading  
import keyboard  
import time  
from datetime import datetime  
  
# Определите требуемую комбинацию клавиш для завершения  
REQUIRED\_KEYS = {'ctrl', 'shift'}  
BAD\_KEYS = {'z', 'x', 'c'}  
  
  
class KeyCombinationTracker:  
 def \_\_init\_\_(self, required\_keys):  
 self.required\_keys = set(required\_keys)  
 self.pressed\_keys = set()  
  
 def add\_key(self, key):  
 self.pressed\_keys.add(key)  
  
 def remove\_key(self, key):  
 self.pressed\_keys.discard(key)  
  
 def is\_combination\_pressed(self):  
 return self.required\_keys.issubset(self.pressed\_keys)  
  
  
class Observer:  
 def \_\_init\_\_(self, subscribers):  
 self.subscribers = subscribers  
  
 def on\_next(self, value, event\_type):  
 for subscriber in self.subscribers:  
 subscriber.on\_next(value, event\_type)  
  
 def on\_completed(self):  
 for subscriber in self.subscribers:  
 subscriber.on\_completed()  
  
 def on\_error(self, e):  
 for subscriber in self.subscribers:  
 subscriber.on\_error(e)  
  
  
class ErrorSubscriber:  
 def \_\_init\_\_(self, thread\_id):  
 self.thread\_id = thread\_id  
 self.error\_count = 0  
  
 def on\_next(self, value, event\_type):  
 pass  
  
 def on\_completed(self):  
 # print(f"ErrorSubscriber {self.thread\_id}: Monitoring completed.")  
 # print(f"Total error count: {self.error\_count}")  
 pass  
  
 def on\_error(self, e):  
 # self.error\_count += 1  
 # print(f"ErrorSubscriber {self.thread\_id}: Monitoring error occurred: {e}")  
 pass  
  
  
class WritingSubscriber:  
 def \_\_init\_\_(self, thread\_id, filename):  
 self.thread\_id = thread\_id  
 self.filename = filename  
 self.key\_count = 0  
  
 def on\_next(self, value, event\_type):  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")  
 log\_entry = f"Thread {self.thread\_id} - {current\_time}: Key {event\_type}: {value}\n"  
 with open(self.filename, 'a') as file:  
 file.write(log\_entry)  
  
 self.key\_count += 1  
 print(log\_entry.strip())  
  
 def on\_completed(self):  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")  
 log\_entry = f"Thread {self.thread\_id} - {current\_time}: detection finished\n"  
 with open(self.filename, 'a') as file:  
 file.write(log\_entry)  
 print(log\_entry.strip())  
  
 def on\_error(self, e):  
 current\_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S.%f")  
 log\_entry = f"Thread {self.thread\_id} - {current\_time}: error detected:\n\t{e}\n"  
 with open(self.filename, 'a') as file:  
 file.write(log\_entry)  
 print(log\_entry.strip())  
  
  
def monitor\_keys(observer, key\_combination\_tracker):  
 is\_running = True  
  
 while is\_running:  
 events = keyboard.read\_event()  
 if events:  
 key = events.name  
 if events.event\_type == keyboard.KEY\_DOWN:  
 event\_type = 'pressed'  
 key\_combination\_tracker.add\_key(key)  
 elif events.event\_type == keyboard.KEY\_UP:  
 event\_type = 'released'  
 key\_combination\_tracker.remove\_key(key)  
 else:  
 continue  
  
 if key\_combination\_tracker.is\_combination\_pressed():  
 is\_running = False  
 observer.on\_completed()  
 continue  
  
 observer.on\_next(key, event\_type)  
  
 if key in BAD\_KEYS and events.event\_type == keyboard.KEY\_DOWN:  
 observer.on\_error(f"Error: Bad key '{key}' pressed.")  
  
  
def main():  
 output\_file = "keys0.txt"  
 with open(output\_file, 'w') as f:  
 f.write('Starting logging:\n')  
  
 writing\_subscriber1 = WritingSubscriber(thread\_id=1, filename=output\_file)  
 writing\_subscriber2 = WritingSubscriber(thread\_id=2, filename=output\_file)  
 error\_subscriber = ErrorSubscriber(thread\_id=3)  
  
 key\_combination\_tracker = KeyCombinationTracker(REQUIRED\_KEYS)  
  
 subscribers = [writing\_subscriber1, writing\_subscriber2, error\_subscriber]  
 observer = Observer(subscribers)  
  
 print(f"Программа запущена. Для завершения нажмите {' + '.join(REQUIRED\_KEYS)}")  
  
 key\_thread = threading.Thread(target=monitor\_keys, args=(observer, key\_combination\_tracker))  
 key\_thread.start()  
  
 key\_thread.join()  
  
 print("Программа завершена.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()